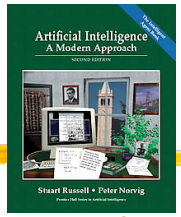
Introduction to Intelligent Systems (co528)

Google “Andy King”

Part	Topic
1	iterative-deepening
2	puzzles
3	blind and informed search
4	minimax
5	constraint programming

Books and resources



- Search Techniques:
  - Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, 2002 (£40 used, library)
  - Judea Pearl, “Heuristics: Intelligent Search Strategies for Computer Problem Solving”, Addison Wesley, 1984 (library)
  - Nils Nilsson, “Problem Solving Methods in Artificial Intelligence”, McGraw-Hill, 1971 (library)
- Constraint Programming:
  - Krzysztof Apt, “Principles of Constraint Programming”, Cambridge, 2003 (£32 used, library)
  - Kim Marriott and Peter Stuckey, “Programming with Constraints”, MIT Press, 1998 (library)
- Java 5 sources available as src.zip from course webpage

Part I

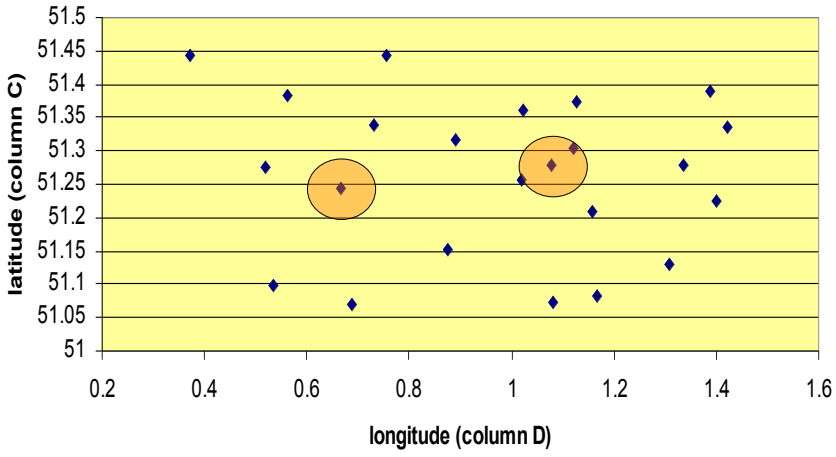
Route finding and iterative deepening

Historical perspective on "Look Ma, no hands" era

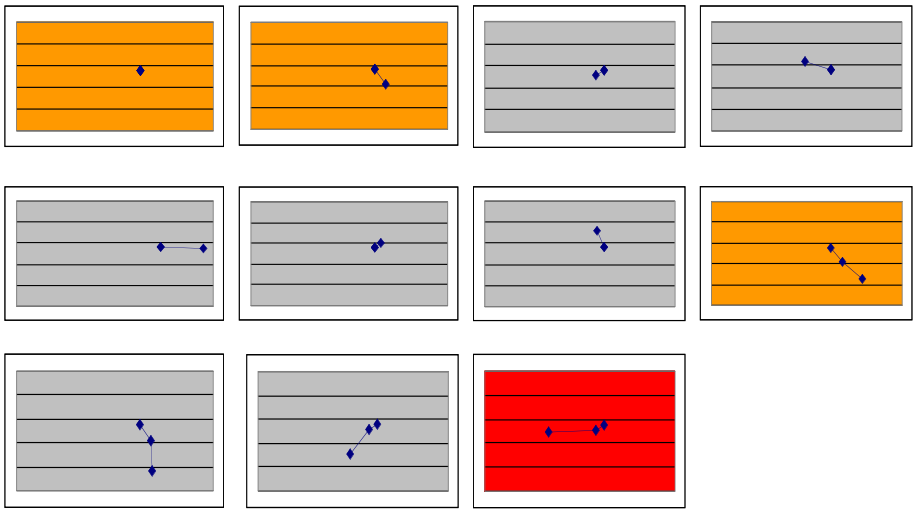
- [Newell and Ernest, *IFIP Congress*, 1965] introduced the phrase "heuristic search"
- [Doran and Michie, *Proceedings of the Royal Society of London*, 294, 1966] developed heuristics for the 8-puzzle and the 15-puzzle
- [Hart, Nilsson and Raphael, *Systems Science and Cybernetics*, SSC-4, 1968] developed A* search; [Erratum in *SIGART*, 1972] (see survey by Nilsson)
- [Pohl, *Machine Intelligence*, 8, 1977] studied relationship between complexity of A* and error in its heuristic (see survey by Pearl)
- [Haralick and Elliot, *Artificial Intelligence*, 14, 1980] developed heuristics for constraint programming (see survey by Apt)

Routefinder for Kent

Latitude and longitude

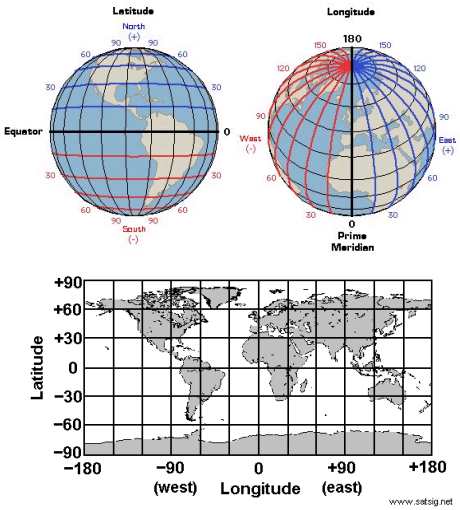


Iterative deepening from Canterbury to Harrietsham



Latitude and longitude details (towns.xls)

ashford	ash	51.1534	0.8746
barham	bar	51.2081	1.1586
canterbury	cant	51.2783	1.0776
chartham	chart	51.255	1.0205
cranbrook	cran	51.0968	0.5354
deal	deal	51.2251	1.3992
dungeness	dung	50.9164	0.9742
dover	dov	51.1295	1.3089
faversham	fav	51.3168	0.89
folkestone	folk	51.0815	1.166
gillingham	gill	51.3839	0.5609
gravesend	graves	51.4419	0.3707
harrietsham	harr	51.244	0.6673
hastings	hast	50.8539	0.5748
herne_bay	hb	51.3735	1.1257
hythe	hy	51.0726	1.0805
maidstone	maid	51.2751	0.5205
margate	mar	51.3893	1.3874
new_romney	nr	50.9859	0.9397
ramsgate	rams	51.3363	1.4211
rye	rye	50.9498	0.7373
sandwich	sand	51.2771	1.337
sheerness	sheer	51.4421	0.756
sittingbourne	sit	51.3378	0.7321
sturry	st	51.3036	1.1211
tenterden	tent	51.0694	0.6891
whitstable	whit	51.3621	1.0223



Town class (infra-structure)

```
public class Town implements Comparable<Town> // key for TreeMap
{   private String name, abbrev;
    private double latitude, longitude;

    Town(String name, String abbrev, double latitude, double longitude)
    {       this.name = name;   this.abbrev = abbrev;
            this.latitude = latitude;   this.longitude = longitude;      }

    public String getName()
    {       return name;    }
    ...
    public int compareTo(Town town) // satisfy Comparable requirements
    {       return name.compareTo(town.getName());  }
    ...
    public String toString()
    {       return abbrev;    }
}
```

graph mapping

```
{ash=[chart, fav, folk, harr, hy, nr, rye, tent],
bar=[cant, dov, folk], cant=[bar, chart, fav, sand,
st, whit], chart=[ash, cant, harr], cran=[hast,
maid], deal=[dov, sand], dov=[bar, deal, folk,
sand], dung=[], fav=[ash, cant, whit], folk=[ash,
bar, dov, hy], gill=[graves, sit], graves=[gill,
maid], harr=[ash, chart, maid], hast=[cran, rye,
tent], hb=[mar, st, whit], hy=[ash, folk, nr], maid=
[cran, graves, harr, sit, tent], mar=[hb, rams, st],
nr=[ash, hy, rye], rams=[mar, sand, st], rye=[ash,
hast, nr, tent], sand=[cant, deal, dov, rams],
sheer=[sit], sit=[gill, maid, sheer], st=[cant, hb,
mar, rams], tent=[ash, hast, maid, rye], whit=[cant,
fav, hb]}
```

Iterative deepening

```
private LinkedList<Town> iterativeDeepening(Town town1, Town town2)
{
      for (int depth = 1; true; depth++)        // doubtful termination
      {
          LinkedList<Town> route = depthFirst(town1, town2, depth);
          if (route != null) return route;          // fast exit
      }
}
```

Depth-limited search

```
private LinkedList<Town> depthFirst(Town last, Town dest, int depth)
{     if (depth == 0) return null;
      else if (last.equals(dest))
      {
              LinkedList<Town> route = new LinkedList<Town>();
              route.add(dest); // construct singleton route
              return route;     }
      else
      {
              LinkedList<Town> nextTowns = graph.get(last);
              for (Town next:nextTowns)       // search top-down
              {    LinkedList<Town> route = depthFirst(next, dest, depth - 1);
                   if (route != null)
                   {     route.addFirst(last);
                         return route;     }
              }
              return null;    }
}
```

Completeness/incompleteness

- A search algorithm is said to be ***complete*** if, given enough resource, it will either:
  - find a route between two points
  - find that no route exists between two points
- A search algorithm that is not complete is said to be ***incomplete***
- Note that an incomplete algorithm may not even terminate for some configurations!

Iterative deepening versus depth-limited search

- Iterative deepening is incomplete since:
  - if no route exists, then deepening will continue *ad infinitum*
- Depth-limited search is complete iff the limit is greater or equal to the length of shortest route between any two points:
  - if no route exists, then search will always detect failure (case 2)
  - if a route exists, then a shortest route exists, thus a route exists who length is smaller or equal to the limit, in which case such a route will be found (case 1)

Recover incompleteness with depth-limited search?

- Recall that depth-limited search is complete if the depth limit exceeds the length of the shortest route between any two points
- Any shortest path cannot contain two points configurations twice, otherwise the path can be shortened:
  - $[c_1, c_2, c_3, c_4, c_3, c_5] \rightarrow [c_1, c_2, c_3, c_5]$
- Thus the length of a shortest path cannot exceed the total number of ***different*** configurations
- Number of configurations is an upper bound on the length of any shortest path

Optimality/sub-optimality

- A search algorithm is said to be ***optimal*** if, given enough resource, it will:
  - always find a shortest route between two points if a route exists between those points
- A search algorithm that is not optimal is said to be ***sub-optimal***
- Note that there may not be a unique shortest route between two configurations

Iterative deepening versus depth-limited search (reprise)

- Suppose the optimality criteria is route length
- Iterative deepening is optimal since:
  - it will only consider a route of length k+1 when all routes of length k have already been considered
- Depth-limited search is sub-optimal because:
  - the only guarantee is that the length of the route will not exceed the limit

Depth-limited without repetition (1 of 2)

```
private LinkedList<Town> depthFirstDevaVu(LinkedList<Town> route,
                                          Town town2, int depth)
{
    if (depth == 0) return null;

    Town last = route.getLast();

    if (last.equals(town2)) return route;
    else
    {
            LinkedList<Town> nextTowns = graph.get(last);
            for (Town next:nextTowns)
                ….

    }
}
```

Depth-limited without repetition (2 of 2)

```
for (Town next:nextTowns)
{
    if (!route.contains(next))
    {
        LinkedList<Town> nextRoute = (LinkedList<Town>) route.clone();
        nextRoute.add(next);
        LinkedList<Town> wholeRoute =
            depthFirstDevaVu(nextRoute, town2, depth - 1);
        if (wholeRoute != null) return wholeRoute;
    }
}
```

Part II

Puzzle solving

(Redundant) representation of a configuration

- System comprises of:
  - 18 bead wheels
  - 11 blue beads
  - 12 yellow beads
  - 11 red beads

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	y	b	b	b	r	r	r	r	r	r	r	b	r	b	b	r	b	y
w2	clockwise	y	b	y	y	y	y	y	y	y	y	y	y	r	r	b	r	r	b

Shifting w1 clockwise by 1

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	b	b	b	r	r	r	r	r	r	r	b	r	b	b	r	b	y	y
w2	clockwise	b	b	y	y	y	y	y	y	y	y	y	y	b	r	b	r	r	b

Final blue/yellow/red configuration

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	y	b	b	b	b	b	b	b	b	b	b	b	y	y	y	y	y	y
w2	clockwise	y	r	r	r	r	r	r	r	r	r	r	r	y	y	y	y	y	y

Generic configurations

- Original and specific puzzles:
  - 18 11/7/4 bead wheels
  - 12 6/4/2 yellow beads
  - 11 7/4/2 blue and red beads
- Generic puzzle:
  - w bead wheels
  - y yellow beads (y even because of vertical symmetry)
  - b = r = w - (y/2 + 1) blue and red beads

	0											r+1					w-1
w1	y	b	b	b	b	b	b	b	b	b	b	y	y	y	y	y	y
w2	y	r	r	r	r	r	r	r	r	r	r	y	y	y	y	y	y

nextConfigs method (1 of 2)

```
private LinkedList<String> nextConfigs(String config)
{ // 1
    String[] wheels = new String[2];
    wheels[0] = config.substring(0, wheel);
    wheels[1] = config.substring(wheel);
    LinkedList<String> result = new LinkedList<String>();
    for (int i = 0; i < 2; i++)
    {  // 2
          for (int j = 1; j < wheel; j++)
          { // 3
              char[] shiftedWheel = new char[wheel];
              for (int k = 0; k < wheel; k++)
                    shiftedWheel[k] = wheels[i].charAt((k+j) % wheel);
              String nextConfig;
```

nextConfigs method (2 of 2)

```
                    String nextConfig;
                    if (i == 0)
                        nextConfig = String.copyValueOf(shiftedWheel) +
                              shiftedWheel[0] +
                              wheels[1].substring(1, blueRed + 1) +
                              shiftedWheel[blueRed + 1] +
                              wheels[1].substring(blueRed + 2);
                    else
                        nextConfig = shiftedWheel[0] +
                              wheels[0].substring(1, blueRed + 1) +
                              shiftedWheel[blueRed + 1] +
                              wheels[0].substring(blueRed + 2) +
                              String.copyValueOf(shiftedWheel);
                    if (!config.equals(nextConfig) && !result.contains(nextConfig))
                        result.add(nextConfig);
                } // 3
          } // 2
    return result;
} // 1
```

On the disequality and inclusion test

- Consider next configurations of rbrb ryyb:

left shift	Wheel 1	both	Wheel 2	both
1	brbr	brbr byyr	yybr	ybrr yybr
2	rbrb	rbrb ryyb	ybry	ybry ybry
3	brbr	brbr byyr	bryy	bbry bryy

- Thus set of only 5 configurations
- 2(w-1) in worse-case as shown by rbbr ryyr:
  - bbrrbyyr, brrbbyyb, rrbbryyb, ybbryyrr, ybbyyrry, rbbyrryy

depthFirst method (1 of 2)

```
private LinkedList<String> depthFirst(String config, String dest, int depth)
{ // 1
      if (depth == 0) return null;
      else if (config.equals(dest))
      {
         LinkedList<String> route = new LinkedList<String>();
         route.add(config);
         return route;
      }
      else
```

depthFirst method (2 of 2)

```
        else
        {
            LinkedList<String> result = nextConfigs(config);
            for (String nextConfig:result)
            {
                  LinkedList<String>
                       route = depthFirst(nextConfig, dest, depth - 1);
                   if (route != null)
                   {
                       route.addFirst(config);
                       return route;
                   }
            }
            return null;
        }
} // 1
```

iterativeDeepening method

```
public LinkedList<String> iterativeDeepening(String config, String dest)
{
      for (int depth = 1; true; depth++)
      {
          System.out.println(depth);
          LinkedList<String> route = depthFirst(config, dest, depth);
          if (route != null) return route;
      }
}
```

Sample runs

```
>java BeadFinder yrryybby ybbyyrry
1
2
3
4
[yrryybby, ryyrrbbr, byybbrrb, ybbyyrry]
>
>java BeadFinder ybbyyrry yrryybby
1
2
3
4
[ybbyyrry, byybbrrb, ryyrrbbr, yrryybby]
```

A problematically long run

```
>java BeadFinder rrbbryyb ybbyyrry
1
2
3
[rrbbryyb, rbbrryyr, ybbyyrry]
>
>java BeadFinder rbrbryyb ybbyyrry
1
2
…
14^C
```

(Time-out after 9 hours on 1.1GHz PC due to incompleteness *or* due to inefficiency?)

Missionaries and cannibals puzzle

- *n* missionaries and *n* cannibals are one side of a river;
- And so is a boat that holds *c* people
- Find the most time efficient way of moving everyone to the other side without leaving a group of missionaries on either side outnumbered by the cannibals

[*Machine Intelligence*, 3, 1968]

What is a configuration?

- Start configuration ⟨n,n,true⟩
  - n missionaries, n cannibals and 1 boat on the initial side
- End configurations ⟨0,0,true⟩ and ⟨0,0,false⟩
  - 0 missionaries and 0 cannibals on initial side
- Consider next configurations for ⟨2,2,true⟩ for the n = 4 and c = 2 instance:
  - ⟨0,2,false⟩ √ cc ~~~ mmmmcc
  - ⟨1,1,false⟩ √ mc ~~~ mmmccc
  - ⟨2,0,false⟩ × mm ~~~ mmcccc

Triple class (1 of 2)

```
public class Triple
{
    private int miss, cann;
    private boolean boat;

    Triple(int miss, int cann, boolean boat)
    {
        this.miss = miss;
        this.cann = cann;
        this.boat = boat;
    }

    public int getMiss()
    {
        return miss;
    }
    // other accessor methods
```

Triple class (2 of 2)

```
    public boolean isValid(int n)
    {
        if (miss == n) return true;                    // case 1
        if (miss == 0) return true;                    // case 2
//        if (miss >= cann && (n - miss) >= (n - cann)) return true;
//        if (miss >= cann && (-miss) >= (-cann)) return true;
//        if (miss >= cann && cann >= miss) return true;
        if (miss == cann) return true;                 // case 3
        return false;
    }

    public String toString()
    {
        return "(" + miss + ", " + cann + ", " + boat + ")";
    }

}
```

nextConfig() (1 of 2)

```
public LinkedList<Triple> nextConfigs(Triple config)
{
        LinkedList<Triple> result = new LinkedList<Triple>();
        for (int moveMiss = 0; moveMiss <= c; moveMiss++)
            for (int moveCann = 0; moveCann <= c - moveMiss; moveCann++)
                if (config.getBoat())
                {
                    int newMiss = config.getMiss() - moveMiss;
                    int newCann = config.getCann() - moveCann;
                    if (newMiss < 0) ;
                    else if (newCann < 0) ;
                    else
                    {
                        Triple triple = new Triple(newMiss, newCann, false);
                        if (triple.isValid(n)) result.add(triple);
                    }
                }
                else
```

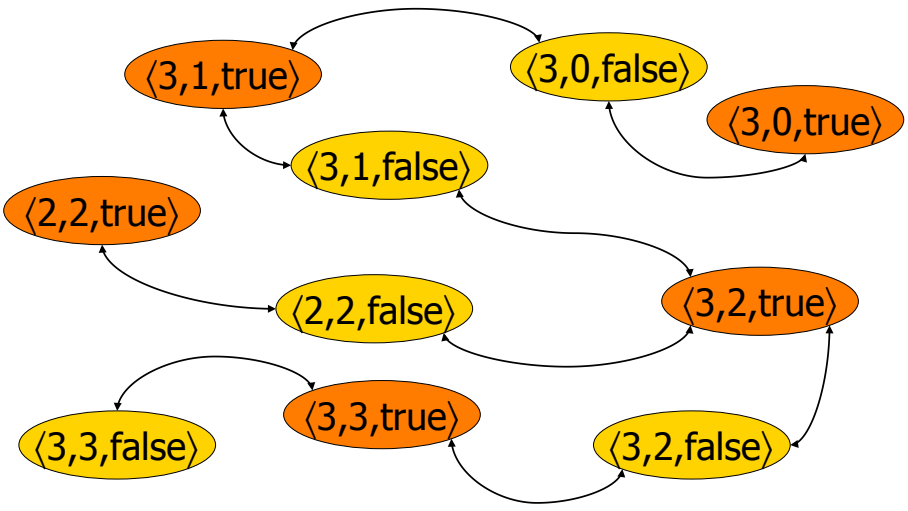
nextConfig() (2 of 2)

```
                else
                {
                    int newMiss = config.getMiss() + moveMiss;
                    int newCann = config.getCann() + moveCann;
                    if (newMiss > n) ;
                    else if (newCann > n) ;
                    else
                    {
                        Triple triple = new Triple(newMiss, newCann, true);
                        if (triple.isValid(n)) result.add(triple);
                    }
                }
        return result;
} // 1
```

Sample runs with iterative deepening

```
>java BoatFinder 5 3
1
...
8
[(5, 5, true), (4, 4, false), (4, 4, true),
(3, 3, false), (3, 3, true), (0, 3, false),
(0, 3, true), (0, 0, false)]
>
>java BoatFinder 3 1
1
2
...
1043 ^C
```

Debugging n = 3 and c = 1



Counting configurations

- Consider n = 3 and initially ignore the boat:
- Cases 1 and 2 both give 4 = n+1 configurations
- Case 3 gives 2 = n-1 additional configurations
- Total of 2(2(n+1) + (n-1)) = 2(3n+1) configurations considering boat positions

mmm			ccc
mmm	c		cc
mmm	cc		c
mmm	ccc		
		mmm	ccc
	c	mmm	cc
	cc	mmm	c
	ccc	mmm	
		mmm	ccc
m	c	mm	cc
mm	cc	m	c
mmm	ccc		

depthLimitedIterativeDeepening method

```
public LinkedList<Triple> depthLimitedIterativeDeepening()
{
      for (int depth = 1; depth < 6*n+2; depth++)
      {
         System.out.println(depth);
         LinkedList<Triple> route = depthFirst(new Triple(n, n, true), depth);
         if (route != null) return route;
      }
      return null;
}
```

Sample runs with depth limited iterative deepening

```
>java BoatFinder 5 3
1
...
8
[(5, 5, true), (4, 4, false), (4, 4, true),
(3, 3, false), (3, 3, true), (0, 3, false),
(0, 3, true), (0, 0, false)]
>
>java BoatFinder 3 1
1
...
19
null
```

Generic concepts

	Route planning	*Bead puzzle*	*Missionaries and cannibals*	*8-puzzle*
state	town	configuration of 2 wheels	who is on what bank	?
operator	road graph	move graph	move graph	?
goal state (s)	destination town	unique target configuration	all people on other side of river	?
branching factor	~4	2(w-1)	~3	~4
solution	journey	series of wheel moves	series of boat and cargo moves	?

Part III

Uninformed versus heuristic search

Uniform-cost search from Canterbury to Harrietsham

0. [(0.0, [cant])]
1. [(5.63, [cant, st]), (6.92, [cant, chart]), (11.59, [cant, whit]), (31.38, [cant, bar]), (28.97, [cant, sand]), (21.40, [cant, fav])]
2. [(6.92, [cant, chart]), (21.40, [cant, fav]), (11.59, [cant, whit]), (31.38, [cant, bar]), (28.97, [cant, sand]), (15.13, [cant, st, hb]), (36.69, [cant, st, mar]), (39.43, [cant, st, rams])]
3. [(11.59, [cant, whit]), (21.40, [cant, fav]), (15.13, [cant, st, hb]), (26.71, [cant, chart, ash]), (28.97, [cant, sand]), (39.43, [cant, st, rams]), (36.69, [cant, st, mar]), (31.38, [cant, bar]), (46.83, [cant, chart, harr])]
4. [(15.13, [cant, st, hb]), (21.40, [cant, fav]), (36.69, [cant, st, mar]), (26.71, [cant, chart, ash]), (24.78, [cant, whit, hb]), (39.43, [cant, st, rams]), (46.83, [cant, chart, harr]), (31.38, [cant, bar]), (27.20, [cant, whit, fav]), (28.97, [cant, sand])]
…

Uniform-cost search from Canterbury to Harrietsham (cont')

16. [(44.58, [cant, st, mar, rams]), (47.47, [cant, bar, folk]), (46.83, [cant, chart, harr]), (50.21, [cant, chart, ash, tent]), (50.21, [cant, fav, whit, hb]), (47.31, [cant, st, rams, mar]), (52.46, [cant, chart, ash, harr]), (52.14, [cant, chart, ash, hy]), (51.18, [cant, st, rams, sand]), (50.37, [cant, bar, dov]), (65.34, [cant, whit, hb, st, mar]), (48.44, [cant, chart, ash, fav]), (48.92, [cant, whit, fav, ash]), (60.51, [cant, chart, ash, folk]), (65.66, [cant, st, hb, whit, fav, ash]), (52.46, [cant, st, hb, mar, rams]), (52.30, [cant, chart, ash, nr]), (57.29, [cant, chart, ash, rye]), (68.07, [cant, whit, hb, st, rams]), (52.95, [cant, sand, dov]), (54.23, [cant, whit, hb, mar]), (74.51, [cant, sand, rams, st]), (66.14, [cant, st, mar, hb]), (66.63, [cant, fav, ash, tent]), (48.60, [cant, sand, rams, mar]), (62.92, [cant, fav, ash, chart]), (76.92, [cant, fav, ash, folk]), (68.88, [cant, fav, ash, harr]), (68.56, [cant, fav, ash, hy]), (68.72, [cant, fav, ash, nr]), (73.71, [cant, fav, ash, rye]), (58.26, [cant, sand, deal, dov])]
17. [(46.83, [cant, chart, harr]), …, (56.33, [cant, st, mar, rams, sand])]

Best-first search from Canterbury to Harrietsham

0. [(45.68, [cant])]

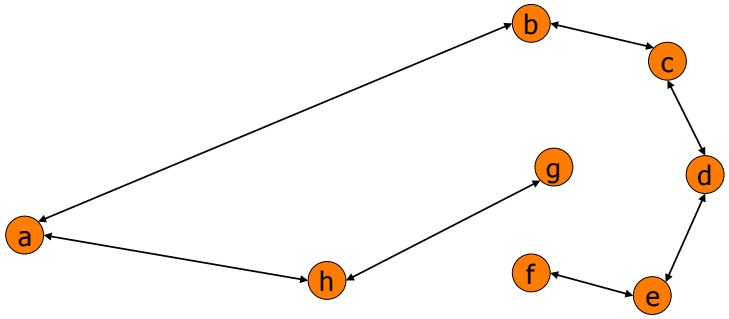
1. [(25.99, [cant, fav]), (50.78, [cant, st]), (39.21, [cant, chart]), (74.39, [cant, sand]), (54.65, [cant, bar]), (41.51, [cant, whit])]

2. [(25.10, [cant, fav, ash]) , (50.78, [cant, st]), (39.21, [cant, chart]), (74.39, [cant, sand]), (54.65, [cant, bar]), (41.51, [cant, whit]), (41.51, [cant, fav, whit])]

3. [(0.00, [cant, fav, ash, harr]), (39.21, [cant, chart]), (19.52, [cant, fav, ash, tent]), (50.78, [cant, st]), (41.63, [cant, fav, ash, nr]), (33.55, [cant, fav, ash, rye]), (41.51, [cant, fav, whit]), (74.39, [cant, sand]), (58.19, [cant, fav, ash, folk]), (54.65, [cant, bar]), (49.63, [cant, fav, ash, hy]), (41.51, [cant, whit]), (39.21, [cant, fav, ash, chart])]

- Route has length of 68.88 kms, hence best-first is sub-optimal (uniform-cost route is 46.83 kms)

Backing out of false paths with best-first search



Backing out of false paths when travelling from a to g

0. [(8.5, [a])]
1. [(2.3, [a,b]), (4.1, [a,h])]
2. [(2.4, [a,b,c]), (4.1, [a,h])]
3. [(2.4, [a,b,c,d]), (4.1, [a,h])]
4. [(2.5, [a,b,c,d,e]), (4.1, [a,h])]
5. [(1.6, [a,b,c,d,e,f]), (4.1, [a,h])]
6. [(4.1, [a,h])]
7. [(0.0, [a,h,g])]

Optimality of uniform-cost

- When a route is expanded, ***all*** routes that are strictly smaller have already been expanded
- Suppose that r is the ***first*** route that is up for expansion that already leads to a goal state
- Route r is a solution but ***assume*** that it is not optimal
- Then a smaller route r′ must exist
- The route r′ would be expanded earlier than r
- Hence r would not be the first route for expansion which leads to a goal state -- a contradiction

Completeness of best-first (and related algorithms)

- Déjà vu check ensures that no town occurs multiply in a route
- Thus each town can occur at most once
- Consider the number of routes possible with just n = 3 towns a, b and c:
  - [a,b,c], [a,c,b], [b,a,c], [b,c,a], [c,a,b], [c,b,a] (3!)
  - [a,b], [b, a], [a, c], [c, a], [b, c], [c, b] (3!)
  - [a], [b], [c] (≤3!)
- Number of different routes is there (≤n(n!)) which is finite whenever n is finite
- Since no route is ever expanded twice, best-first will either:
  - Terminate by expanding all routes without finding a solution (case 2)
  - Terminate earlier by finding a solution (case 1)
- Therefore best-first search is complete

Pair class (1 of 2)

```
import java.util.*;

public class Pair implements Comparable<Pair>
{
    private double rank;
    private LinkedList<Town> route;

    public double getRank()
    {
        return rank;
    }

    public LinkedList<Town> getRoute()
    {
        return route;
    }
```

Pair class (2 of 2)

```
    Pair(double rank, LinkedList<Town> route)
    {
        this.rank = rank;
        this.route = route;
    }

    public int compareTo(Pair pair)
    {
        if (rank > pair.getRank()) return 1;
        else if (rank < pair.getRank()) return -1;
        else return 0;              // return (int) (rank - pair.getRank());
    }

    public String toString()        // force short debug traces
    {
        return "(" + String.format("%.2f", rank) + ", " + route + ")";
    }
}
```

Uniform-cost and best-first methods (1 of 2)

```
   private LinkedList<Town> uniformCost(Town town1, Town town2)
   {
         LinkedList<Town> route = new LinkedList<Town>();
         route.add(town1);
         PriorityQueue pairs = new PriorityQueue();
         pairs.add(new Pair(0.0, route)); // uniform-cost
//       pairs.add(new Pair(estimateDistance(town1, town2), route)); // best
         while (true)
         {
//          System.out.println(pairs);              // debug traces
            if (pairs.size() == 0) return null;     // no solutions exist
            Pair pair = (Pair) pairs.poll();        // retrieve and remove (log)
            route = pair.getRoute();
            Town last = route.getLast();
            if (last.equals(town2)) return route;   // exit loop with solution
           LinkedList<Town> nextTowns = graph.get(last);
            for (Town next:nextTowns)
```

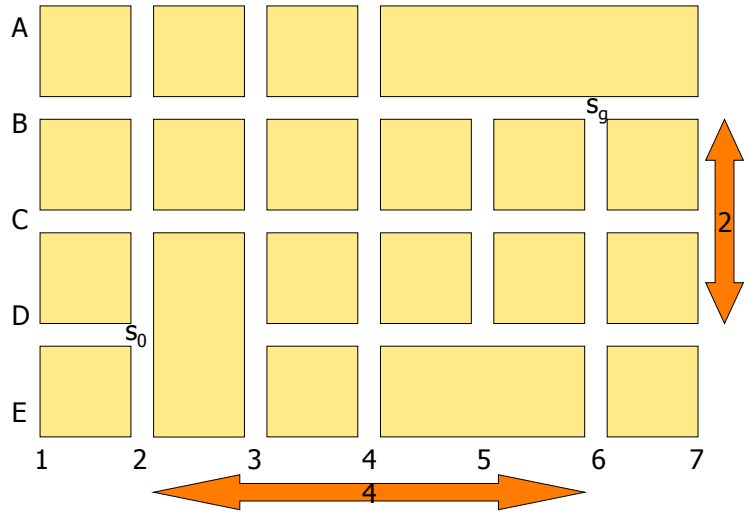
Uniform-cost and best-first methods (2 of 2)

```
         for (Town next:nextTowns)
         {
            if (!route.contains(next))                          // deja vu
            {
               LinkedList<Town> nextRoute = new LinkedList<Town>(route);
               nextRoute.addLast(next);
               double distance = actualDistance(nextRoute);     // uniform
//             double distance = estimateDistance(next, town2); // best-first
               pairs.add(new Pair(distance, nextRoute));        // log too
            }
         }
      }
}
```

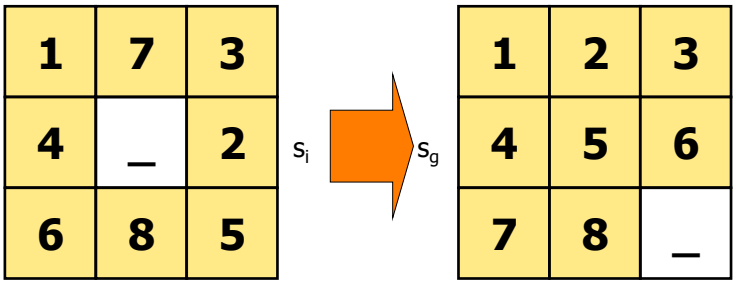
Heuristics in AI

- Greeks: "heuriskekein" means to "to find" (so Archimedes shouted "Heureka")
- 60s: heuristic as opposed to algorithmic: "a process that may solve a given problem, but offers no guarantee of doing so", [Newell and Simon, *Lernende Automaten* (Automata), 1963]
- 70s: heuristic programming used for expert systems/rule-based programming in which "rules of thumb" were extracted from domain experts
- 80s: a process that improves average-case performance of an ***algorithm*** but does not necessarily improve worse-case performance

Straight-line distance = sqrt(20) but City block distance = 6

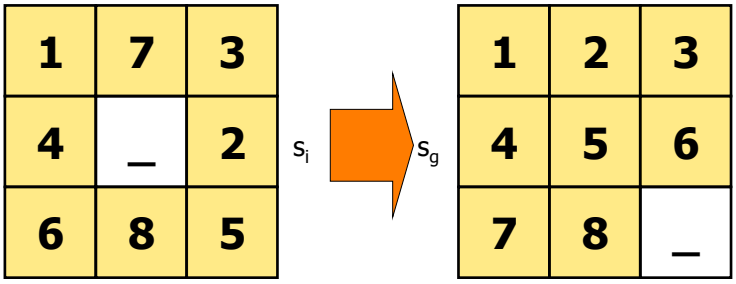


Heuristic for 8-puzzle (not needed for the assessment)



- Tiles 7, 2, 6 and 5 in s_i are out-of-place (the blank is not a tile)
- Each move can correct at most of one tile
- Thus an admissible heuristic is the total number of tiles that remain out-of-place (4 for s_i)

Better heuristic for 8-puzzle



- Each move can move at ***most of one tile one position nearer*** its destination
- Tile 7 requires 1 horizontal and 2 vertical moves
- Minimum of (1+2) + (1+1) + (2+1) + (1+1) = 3+2+3+2 = 10 moves required for s_i

Evaluation (ranking) functions

- Evaluation function $e(r, s_g)$ takes a route r and a goal state s_g and gives a number that represents the desirability of expanding r
- For uniform-cost search: $e(r, s_g) = g(r)$ where g(r) represents the cost of r
- For best-first search: $e(r, s_g) = h(r, s_g)$ where h is a heuristic that estimates the cost of travelling on from last state in r to s_g
- For A* search: $e(r, s_g) = g(r) + h(r, s_g)$ [with a technical caveat on $h(r, s_g)$]

A* search from Canterbury to Harrietsham

$e(r, s_g) = g(r) + h(r, s_g)$ is an estimate of cost of a complete journey that progresses along the route r and then continues onto s_g

0. [(45.68, [cant])]

1. [(46.13, [cant, chart]) , (56.41, [cant, st]), (47.40, [cant, fav]), (103.36, [cant, sand]), (86.03, [cant, bar]), (53.10, [cant, whit])]

2. [(46.83, [cant, chart, harr]), (56.41, [cant, st]), (47.40, [cant, fav]), (103.36, [cant, sand]), (86.03, [cant, bar]), (53.10, [cant, whit]), (51.81, [cant, chart, ash])]

A* method (1 of 2)

```
private LinkedList<Town> aStar(Town town1, Town town2)
{
        LinkedList<Town> route = new LinkedList<Town>();
        route.add(town1);
        PriorityQueue pairs = new PriorityQueue();
        pairs.add(new Pair(estimateDistance(town1, town2), route)); // A*

        while (true)
        {
//          System.out.println(pairs);          // debug traces
            if (pairs.size() == 0) return null;  // no solutions exist
            Pair pair = (Pair) pairs.poll();     // retrieve and remove (log)
            route = pair.getRoute();
            Town last = route.getLast();
            if (last.equals(town2)) return route;  // exit loop with solution
            LinkedList<Town> nextTowns = graph.get(last);
            for (Town next:nextTowns)
```

A* method (2 of 2)

```
        for (Town next:nextTowns)
        {
            if (!route.contains(next))                    // deja vu
            {
                LinkedList<Town> nextRoute = new LinkedList<Town>(route);
                nextRoute.addLast(next);
                double distance = actualDistance(nextRoute);   // A*
                distance += estimateDistance(next, town2);     // A*
                pairs.add(new Pair(distance, nextRoute));      // log too
            }
        }
    }
}
```

Expansion counts (efficiency)

search problem		*unif*	*best*	*A**	*length*
canterbury	gillingham	297	6	32	109.75
dover	tenterden	48	3	5	74.19
maidstone	dungeness	78627	78627	78627	none
deal	faversham	25	3	3	65.02
sheerness	cranbrook	6	3	3	65.66
sittingbourne	sandwich	72	7	7	118.12
ashford	ramsgate	56	4	4	66.14
new_romney	whitstable	23	4	5	62.92
barham	gravesend	464	6	16	121.34

Technical caveat revealed

- A* is a search algorithm that uses the evaluation function $e(r, s_g) = g(r) + h(r, s_g)$ where $h(r, s_g)$ is admissible
- A heuristic $h(r, s_g)$ is admissible if and only if:
  - it does ***not over-estimate*** the distance from the end of the route r to the goal state s_g
- Examples of admissible heuristics:
  - straight-line distance (using polar radius a for r)
  - Manhattan block distance
  - both 8-puzzle move heuristics

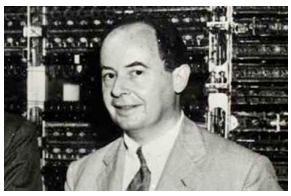
Optimality of A*

- Suppose the start state is s_0, s_g is the ***single*** goal state and $r = [s_0, s_1, \ldots, s_j, s_g]$ is ***an*** optimal route from s_0 to s_g
- ***Assume*** A* returns $r' = [s_0, s'_1, \ldots, s'_k, s_g]$ and ***g(r) < g(r')***
- Let $j = \max\{ n \mid [s_0, s_1, \ldots, s_n] = [s_0, s'_1, \ldots, s'_n]\}$
- Path $[s_0, s_1, \ldots, s_{j+1}]$ was never expanded by A* thus:
  - $g([s_0, s'_1, \ldots, s'_{j+1}]) + h(s'_{j+1}) \leq g([s_0, s_1, \ldots, s_{j+1}]) + h(s_{j+1})$
  - $g([s_0, s'_1, \ldots, s'_{j+1}, s'_{j+2}]) + h(s'_{j+2}) \leq g([s_0, s_1, \ldots, s_{j+1}]) + h(s_{j+1})$
  - …
  - $g([s_0, s'_1, \ldots, s'_{j+1}, s'_{j+2}, \ldots, s_g]) + h(s_g) \leq g([s_0, s_1, \ldots, s_{j+1}]) + h(s_{j+1})$
- Since h is ***admissible***, $g([s_0, s_1, \ldots, s_{j+1}]) + h(s_{j+1}) \leq g(r)$
- Therefore $g([s_0, s'_1, \ldots, s'_{j+1}, s'_{j+2}, \ldots, s_g]) + h(s_g) \leq g(r)$
- Thus $g(r') + h(s_g) \leq g(r)$ hence $g(r') \leq g(r)$ which is a ***contradiction***
- Thus $g(r) \geq g(r')$ and since r is optimal it follows ***g(r)=g(r')***

Summary statement

	optimal	*complete*	*efficient*
breadth-first	depends	yes	no
uniform-cost	yes	yes	no
best-first	no	yes	yes
A*	yes	yes	yes

Part IV

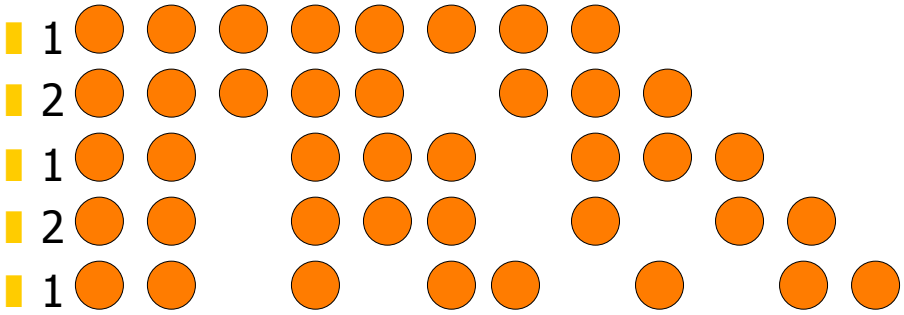


Two player games, Minimax and Alpha-beta pruning

The Game of Jianshizi (Nim)

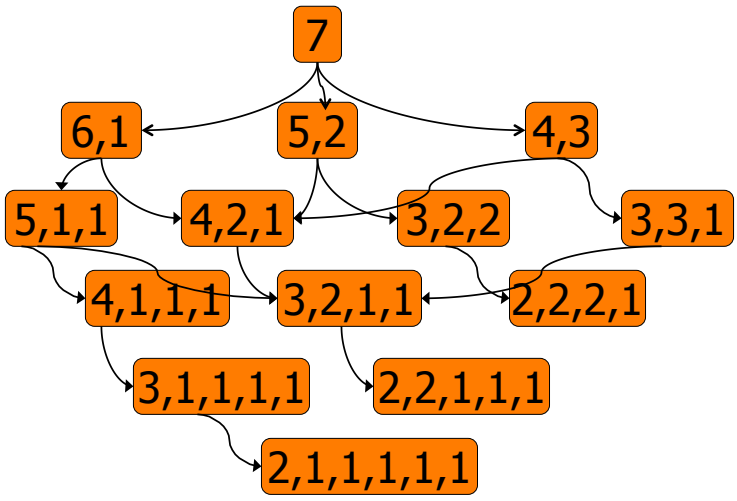
- There are two opponents
- A number of stones are placed on a table
- At each move the player must divide a pile of stones into two non-empty piles of different sizes
- Nim is a so-called misère (povery) game in that the first person who cannot move looses

Example game play for 8 stones



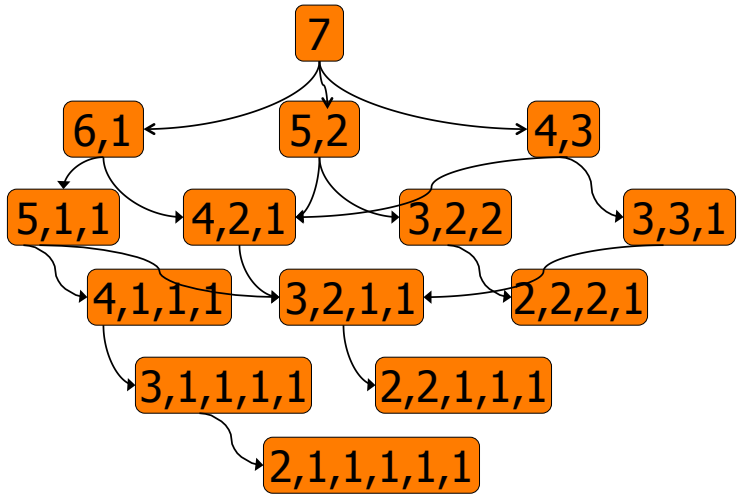
- Player 2 then wins

Every game play for 7 stones



Max = Maximus (player) vs Min = Mina (opponent)

- Max
- Min
- Max
- Min
- Max
- Min

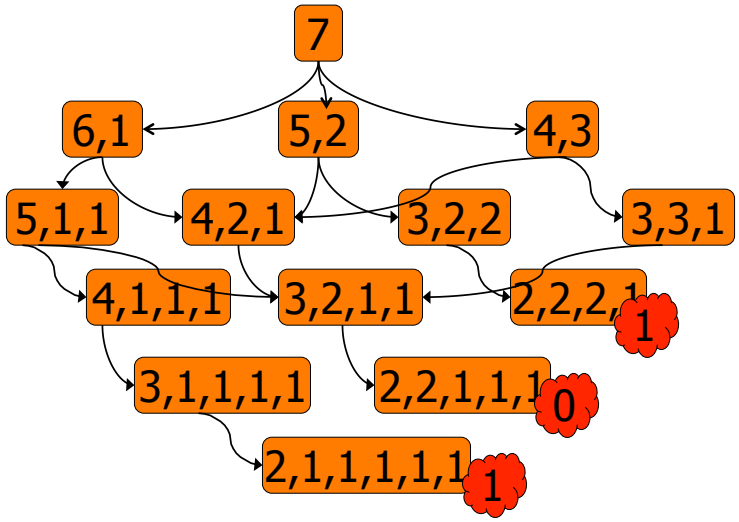


Minimax [Von Neumann, 1928] for reasoning about who wins

- Max wants to maximise his score
- Min wants to minimise Max's score
- Decorate the leaf nodes with a value that indicates who wins:
  - ie. 1 if Max wins
  - ie. 0 if Max loses
- Decorate the nodes of the tree by propagating values upwards

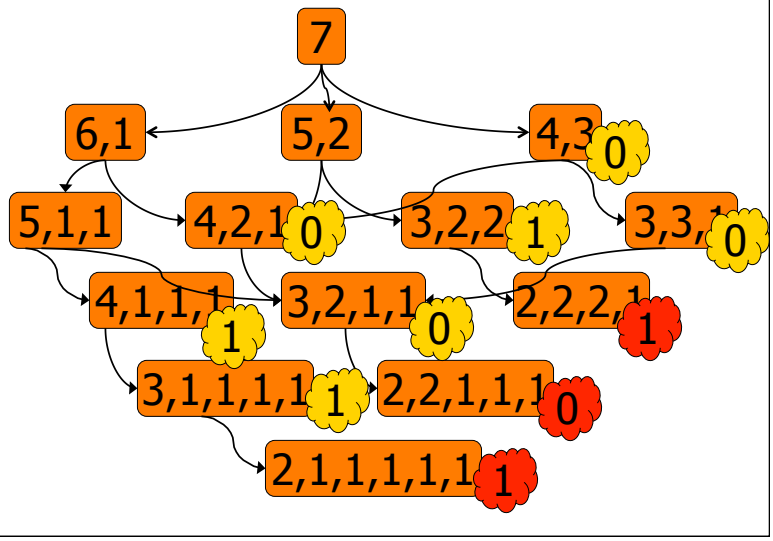
Decorating terminal nodes

- Max
- Min
- Max
- Min
- Max
- Min



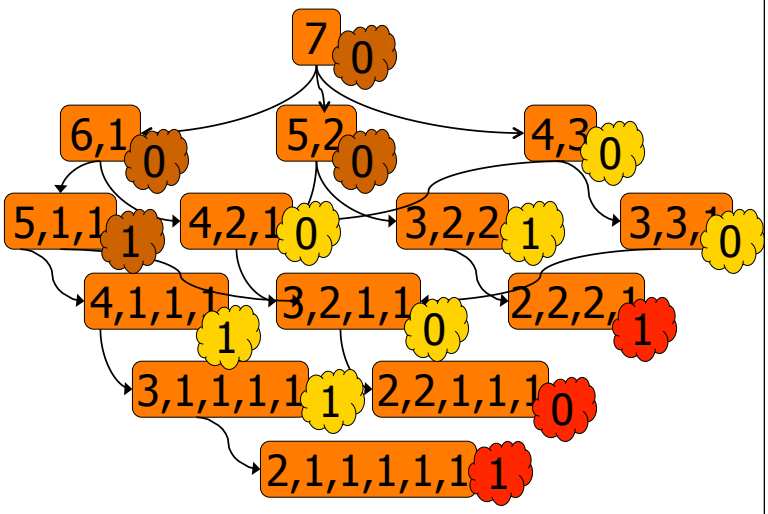
Decorate nodes with just one successor

- Max
- Min
- Max
- Min
- Max
- Min



Can a player win *if* they make good choices? (Minimax)

- Max
- Min
- Max
- Min
- Max
- Min



Reviewing the complete search tree

- What are Max's prospects for winning?
- What move should Min do if she has piles of 5, 2 stones?
- Suppose Min makes the wrong move into 5, 1, 1 stones. What should Max do next?

- The decoration represents the best that the player (in that state) can achieve

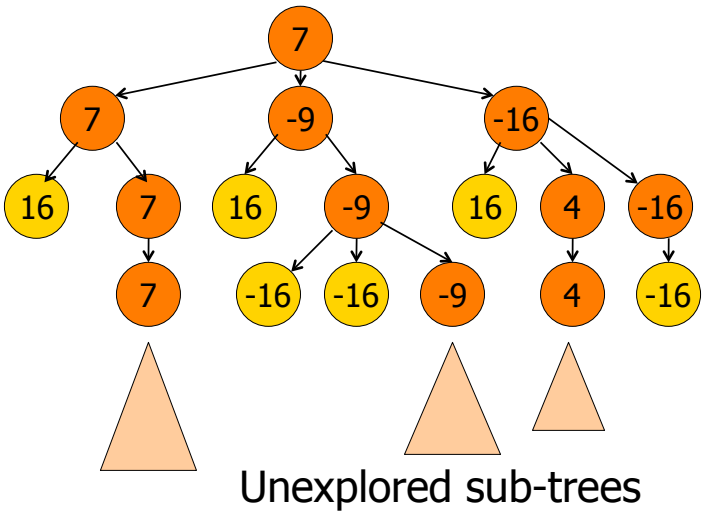
Other games



- Theoretically perfect: 0-1 games where one player has as much information as their opponent (Go, Othello)
- Theoretically imperfect: poker
- Practically imperfect: cannot see to game end
- At a Max ply in Chess:
  - ie. if Max checkmates Min, then value of +32
  - ie. if Min checkmates Max, value of -32
  - ie. else heuristic strength of -32 ≤ value ≤ 32

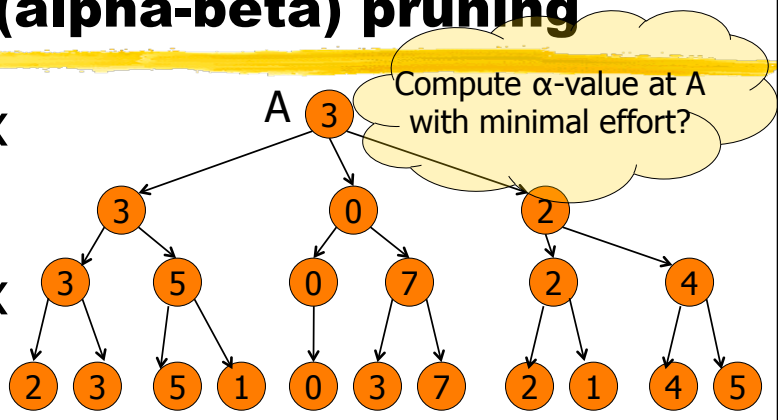
Heuristic Minimax for Chess with 4-ply look ahead

- Max
- Min
- Max
- Min



α–β (alpha-beta) pruning

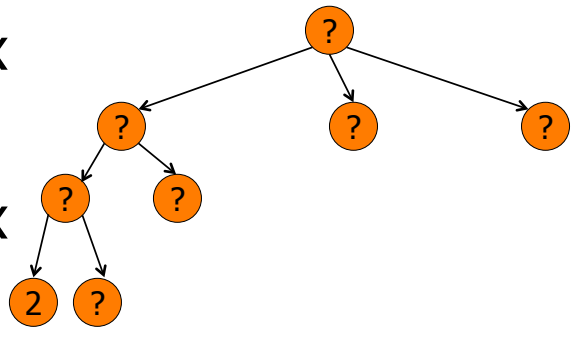
- Max
- Min
- Max
- Min



- α -values in Max ply never decrease when another child is known
- β -values in Min ply never increase

α–β pruning (step 1)

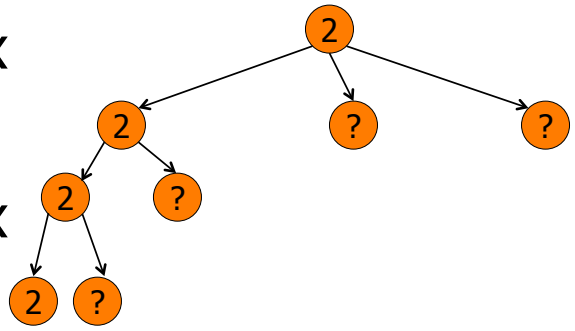
- Max
- Min
- Max
- Min



Descend to depth of 4-ply and calculate heuristic of 2, say, for leaf

α–β pruning (step 2)

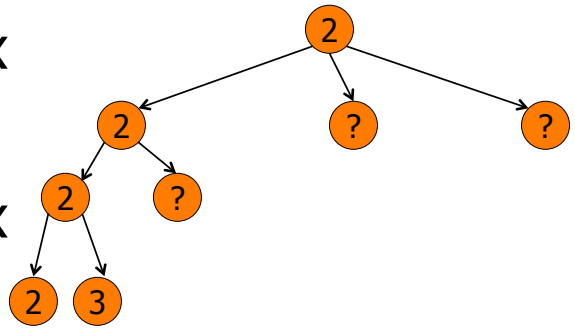
- Max
- Min
- Max
- Min



Propagate heuristic value up to root

α–β pruning (step 3)

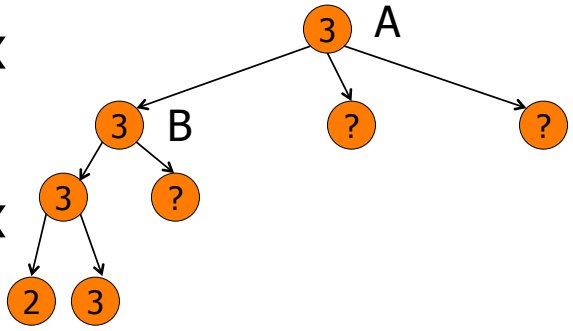
- Max
- Min
- Max
- Min



Descend and calculate next heuristic value, say, 3

α–β pruning (step 4)

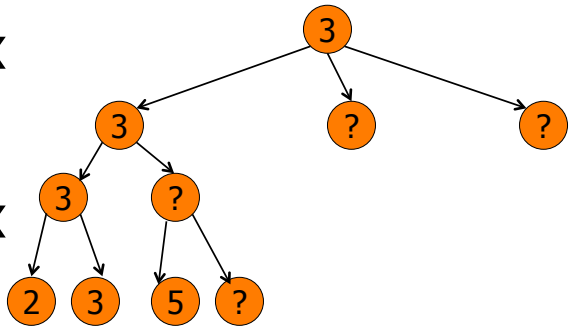
- Max
- Min
- Max
- Min



Propagate 3 up to root
Update A and B since they only have one evaluated successor

α–β pruning (step 5)

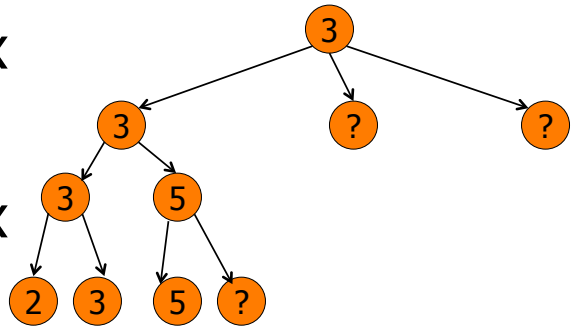
- Max
- Min
- Max
- Min



Descend to depth of 4-ply and assign heuristic of 5, say, to next leaf

α–β pruning (step 6)

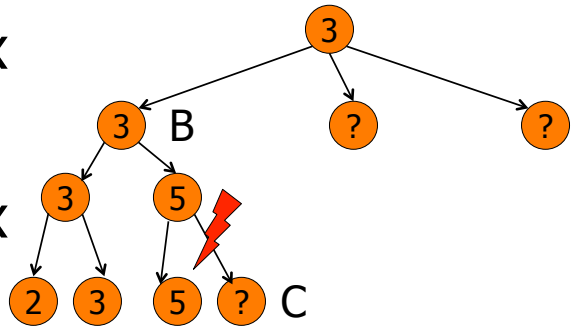
- Max
- Min
- Max
- Min



Propagate 5 back up to root

α–β pruning (step 7)

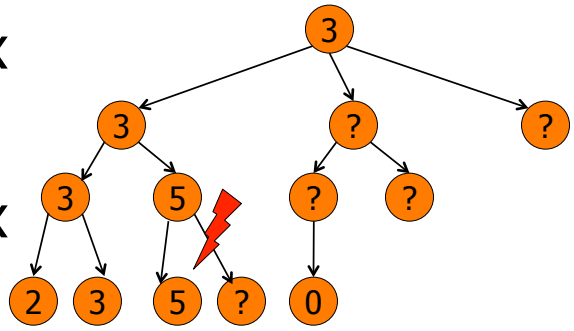
- Max
- Min
- Max
- Min



No need to compute heuristic for C
since this cannot effect B

α–β pruning (step 8)

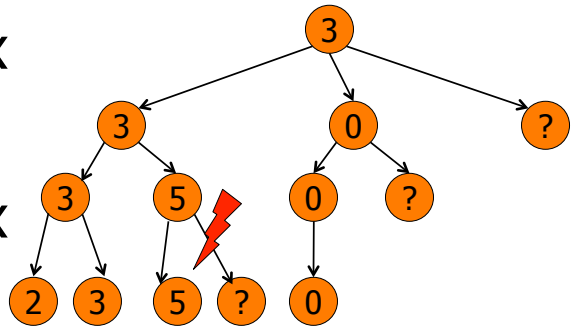
- Max
- Min
- Max
- Min



Descend to depth of 4-ply and assign heuristic of 0, say, to next leaf

α–β pruning (step 9)

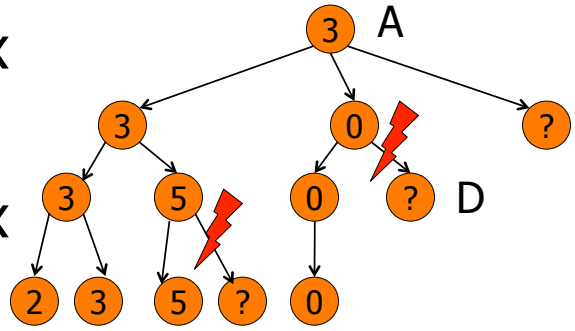
- Max
- Min
- Max
- Min



Propagate 0 up the tree

α–β pruning (step 10)

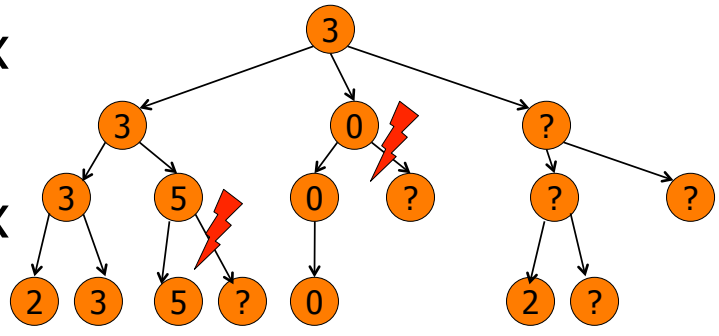
- Max
- Min
- Max
- Min



No need to compute heuristic for D since this cannot effect A

α–β pruning (step 11)

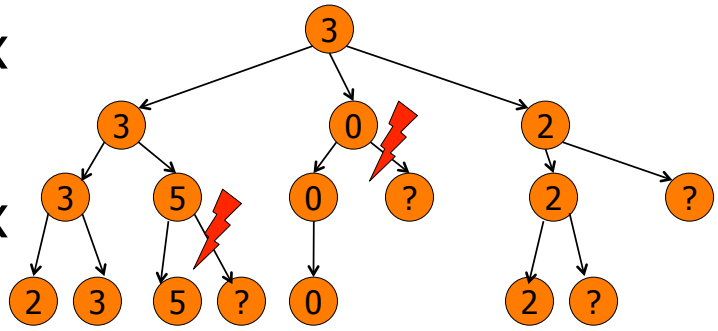
- Max
- Min
- Max
- Min



Descend to depth of 4-ply and assign heuristic of 2, say, to next leaf

α–β pruning (step 12)

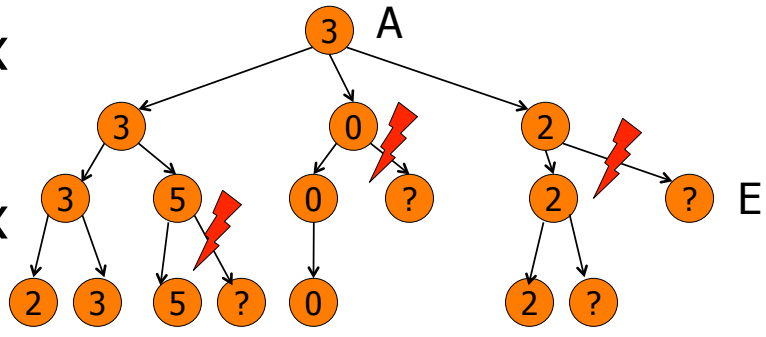
- Max
- Min
- Max
- Min



Propagate 2 up the tree

α–β pruning (step 13)

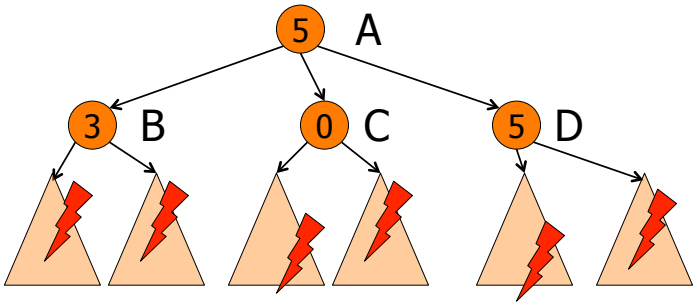
- Max
- Min
- Max
- Min



No need to compute heuristic for D
since this cannot effect A

How can Max make use of α–β pruning when at A?

- Max
- Min
- Max
- Min



Find β-values at B, C and D
Move to D as best chance of winning

Part V

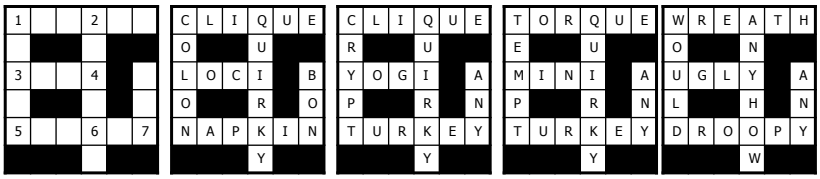
Constraint programming

The crossword compiling problem (fill-in crossword)

- Given a dictionary of words:

torque	colon	tempt	bon	mini	pique
quirky	quay	any	encore	turkey	rue
clique	droopy	crypt	anyhow	yogi	would
loci	wreath	napkin	ugly		

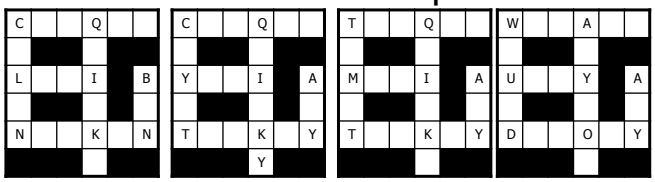
- and a crossword grid:



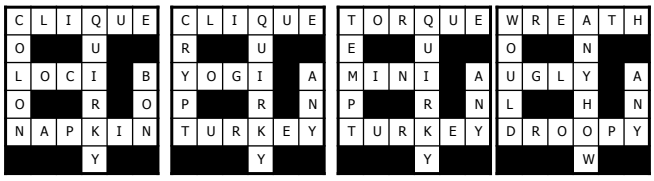
- Insert a subset of the dictionary (possibly using a word twice) into the vertical and horizontal word positions

Crossword compiling in 2 steps

- It is sufficient to find all combinations of characters that can arise at the intersection points:

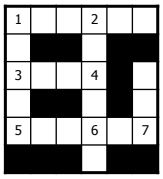


- Words can then be fleshed out by searching the dictionary like so (may generate more solutions):



Arc-consistency algorithm (initialisation and pass 1)

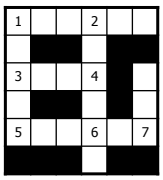
s1 = … = s7 = {a, b, …, z}
w = {1a, 3a, 5a, 1d, 2d, 7u}



consider 1a ∈ w
w := w - {1a}
d' := { w' ∈ d | |w'| = 6 ∧ w'[1] ∈ s1 ∧ w'[4] ∈ s2}
= {torque, quirky, encore, turkey,
clique, droopy, anyhow, wreath, napkin}
s1 := {w' [1] | w' ∈ d' } = {t, n, c, q, a, e, d, w}
s2 := {w' [4] | w' ∈ d' } = {k, q, r, h, o, a}
w := w ∪ {1d}
w := w ∪ {2d}

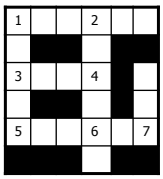
Arc-consistency algorithm (pass 2)

consider 3a ∈ w
w := w - {3a}
d' := {w' ∈ d | |w'| = 4 ∧ w'[1]∈s3 ∧ w'[4]∈s4}
= {mini, quay, yogi, loci, ugly}
s3 := {u, y, q, l, m}
s4 := {y, i}
w := w ∪ {1d}
w := w ∪ {2d}



Arc-consistency algorithm (pass 3)

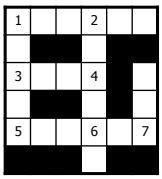
consider 5a ∈ w
w := w - {5a}
d' := {torque, quirky, encore, turkey,
clique, droopy, anyhow, wreath, napkin}
s5 := {t, n, c, q, a, e, d, w}
s6 := {k, q, r, h, o, a}
s7 := {y, n, e, w, h}
w := w ∪ {1d}
w := w ∪ {2d}
w := w ∪ {7u}



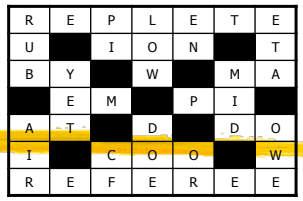
Arc-consistency algorithm (pass 4 2 more needed)

consider 1d ∈ w
w := w - {1d}
d' := {colon, tempt,
picque, crypt, would}
s1 := {t, c, w}
s3 := {m, y, u, l}
s5 := {t, d, n}
w := w ∪ {1a}
w := w ∪ {3a}
w := w ∪ {5a}

w	{1d, 2d, 7u}
s1	{t, n, c, q, a, e, d, w}
s2	{k, q, r, h, o, a}
s3	{u, y, q, l, m}
s4	{y, i}
s5	{t, n, c, q, a, e, d, w}
s6	{k, q, r, h, o, a}
s7	{y, n, e, w, h}

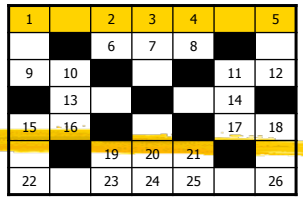


Weak pruning/many iterations for 75,706 words?



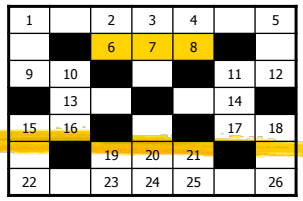
s1	abcdefghijklmnopqrstuvwxyzàéêè	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzàéêè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyzàéêè	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyzàéêè	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghijklmnopqrstuvwxyzàéêè	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyzàéêè	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghijklmnopqrstuvwxyzàéêè	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghijklmnopqrstuvwxyzàéêè	s21	abcdefghijklmnopqrstuvwxyzàéêè
S9	abcdefghijklmnopqrstuvwxyzàéêè	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefghijklmnopqrstuvwxyzàéêè	s23	abcdefghijklmnopqrstuvwxyzàéêè
S11	abcdefghijklmnopqrstuvwxyzàéêè	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefghijklmnopqrstuvwxyzàéêè	s25	abcdefghijklmnopqrstuvwxyzàéêè
S13	abcdefghijklmnopqrstuvwxyzàéêè	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 1



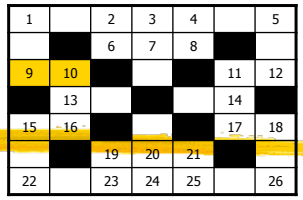
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyzàéêè	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghijklmnopqrstuvwxyzàéêè	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghijklmnopqrstuvwxyzàéêè	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxyzàéêè	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefghijklmnopqrstuvwxyzàéêè	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxyzàéêè	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefghijklmnopqrstuvwxyzàéêè	s25	abcdefghijklmnopqrstuvwxyzàéêè
S13	abcdefghijklmnopqrstuvwxyzàéêè	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 2



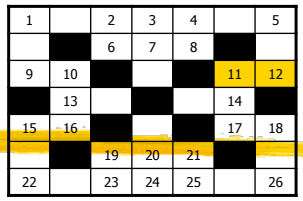
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxyzàéêè	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefghijklmnopqrstuvwxyzàéêè	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxyzàéêè	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefghijklmnopqrstuvwxyzàéêè	s25	abcdefghijklmnopqrstuvwxyzàéêè
S13	abcdefghijklmnopqrstuvwxyzàéêè	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 3



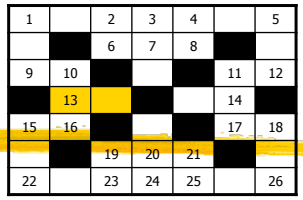
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxyzàéêè	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefghijklmnopqrstuvwxyzàéêè	s25	abcdefghijklmnopqrstuvwxyzàéêè
S13	abcdefghijklmnopqrstuvwxyzàéêè	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 4



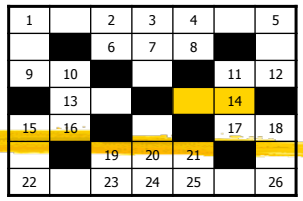
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxyzàéêè	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 5



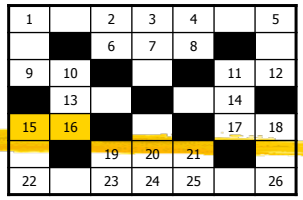
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefghijklmnopqrstuvwxyzàéêè
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 6



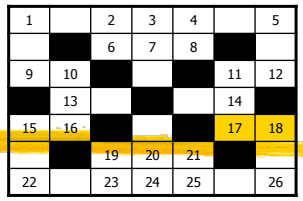
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxyzàéêè
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefghijklmnopqrstuvwxyzàéêè
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 7



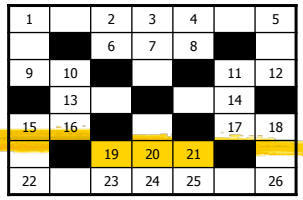
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxyzàéêè
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefghijklmnopqrstuvwxyzàéêè
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 8



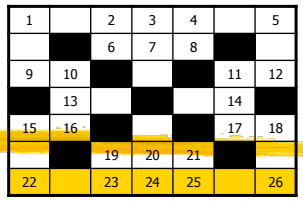
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyzàéêè
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghijklmnopqrstuvwxyzàéêè
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghijklmnopqrstuvwxyzàéêè
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 9



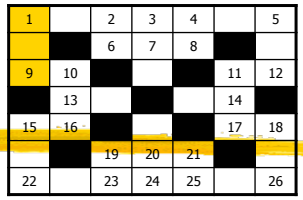
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyzàéêè
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzàéêè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyzàéêè
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyzàéêè
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghijklmnopqrstuvwxyzàéêè

Pass 10



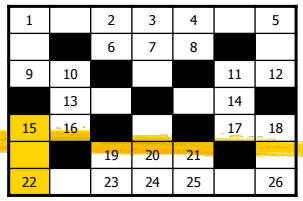
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghijklmnopqrstuvwxy	s22	abcdefghijklmnopqrstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 11



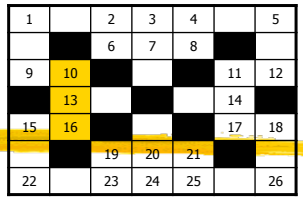
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghijklmnopqrstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 12



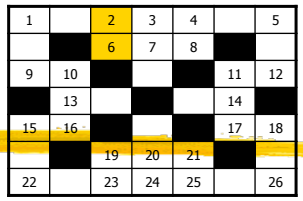
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghijklmnopqrstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 13



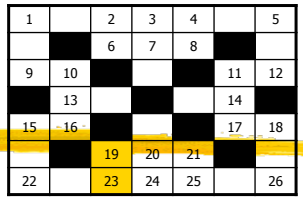
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxyzè	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefghijklmnopqrstuvwxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 14



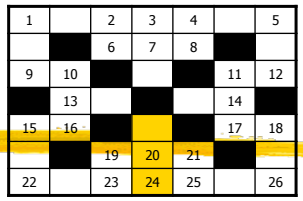
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxyz
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefghijklmnopqrstuvwxyzè
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 15



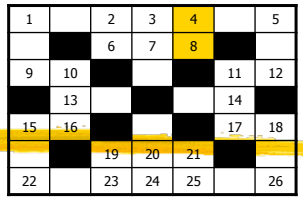
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghijklmnopqrstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 16



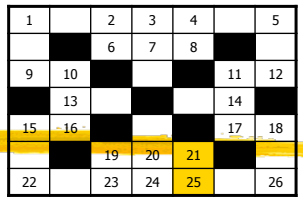
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxyz	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefghiklmnoprstuvwxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 17



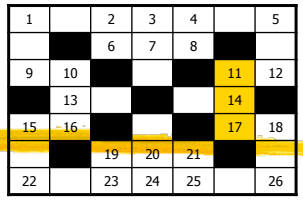
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxy	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefhilmnoprstvxyz	s21	abcdefghiklmnoprstuvwxyz
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefghijklmnopqrstuvwxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 18



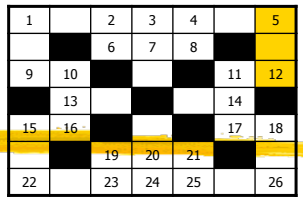
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxyz
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxy	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefhilmnoprstvxyz	s21	abcdefghiklmnoprstuvwxy
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnopqrstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefhilmnoprstvxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 19



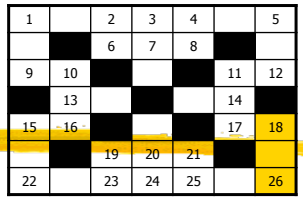
s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxy
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxy	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyzé	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefhilmnoprstvxyz	s21	abcdefghiklmnoprstuvwxy
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnoprstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefhilmnoprstvxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

Pass 20



s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxy
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxy	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyz	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefhilmnoprstvxyz	s21	abcdefghiklmnoprstuvwxy
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnoprstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefhilmnoprstvxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyzé

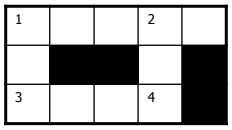
Pass 21 (final)



s1	abcdefghijklmnopqrstuvwxyz	s14	abcdefhilmnoprstvxy
s2	abcdefghijklmnopqrstuvwxy	s15	abcdefghijklmnopqrstuvwxy
s3	abcdefghijklmnopqrstuvwxyz	s16	abcdefhilmnoprstvxyz
s4	abcdefghijklmnopqrstuvwxy	s17	abcdefghijklmnopqrstuvwxy
s5	abcdefghiklmnoprstuwxyz	s18	abcdefhilmnoprstvxyz
s6	abcdefhilmnoprstvxyz	s19	abcdefghijklmnopqrstuvwxy
s7	abcdefghiklmnoprstuvwxy	s20	abcdefghiklmnoprstuvwxy
s8	abcdefhilmnoprstvxyz	s21	abcdefghiklmnoprstuvwxy
s9	abcdefghiklmnoprstuvwxy	s22	abcdefghiklmnoprstuvwxyz
s10	abcdefhilmnoprstvxyz	s23	abcdefhilmnoprstvxyz
s11	abcdefghijklmnoprstuvwxy	s24	abcdefghiklmnoprstuvwxyz
s12	abcdefhilmnoprstvxyz	s25	abcdefhilmnoprstvxyz
s13	abcdefghiklmnoprstuvwxy	s26	abcdefghiklmnoprstuwxyz

Locating a solution

- Consider the following grid and dictionary:



bon	rue
elle	noon
beers	rambo

s1	{b, r}
s2	{b, r}
s3	{e, n}
s4	{e, n}

- But there is no solution to this problem:

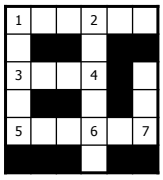
B	E	E	R	S
O			?	
N	O	O	N	

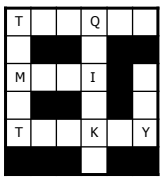
R	A	M	B	O
U			?	
E	L	L	E	

- A solution does not exist ***if*** one intersection is empty
- A solution does not ***necessarily*** exist if all the intersections are non-empty
- A solution exists ***if*** all the intersections are singletons

Search and Arc-consistency

step	pass 6	search	1a	1d	2d	5a
s1	{t,c,w}	{t}	{t}	{t}	{t}	{t}
s2	{q,a}	{q,a}	{q}	{q}	{q}	{q}
s3	{m,y,u,l}	{m,y,u,l}	{m,y,u,l}	{m}	{m}	{m}
s4	{y,i}	{y,i}	{y,i}	{y,i}	{i}	{i}
s5	{t,d,n}	{t,d,n}	{t,d,n}	{t}	{t}	{t}
s6	{k,o}	{k,o}	{k,o}	{k,o}	{k}	{k}
s7	{n,y}	{n,y}	{n,y}	{n,y}	{n,y}	{y}



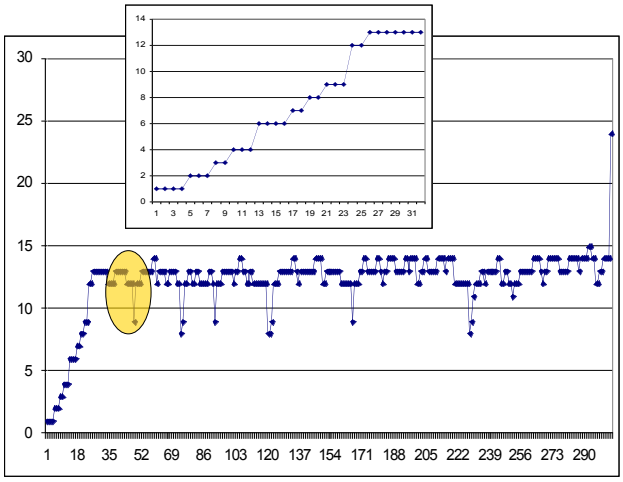


Non-example of backtracking

s1	crsbzhfedtlam	s4	ibcpsf
s1	crsbzh	s4	ibc
s1	crs	s4	i
s1	c	…	…
s2	asurnolpimb	…	…
s2	asurn	s58	p
s2	as	s59	l
s2	a	s60	o
s3	tmhsyre	s62	cwdpmh
s3	tmh	s62	cwd
s3	t	s62	c
s4	ibcpsftarogvw	s63	g

- Grid:
  - 40 words
  - 64 intersections
- Dictionary:
  - 75,706 English words
- Timing:
  - 23 seconds
  - 3.2GHz
  - 1 GByte RAM

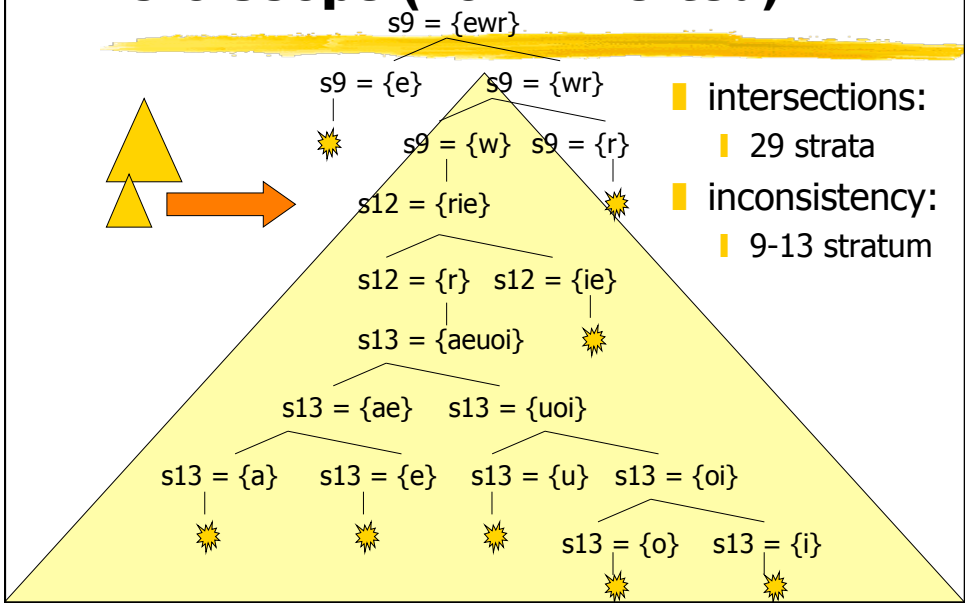
Example of backtracking (level i where si was last halved)



	c	r	a	f	t
c	h	i	l	l	i
r	a	g	l	a	n
a	s	h	o	r	e
b	e	t	t	e	d

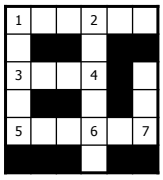
- Dictionary:
  - 75,706 words
- Timing:
  - 83 seconds

Backtracking under the micro-scope (now inverted)



- intersections:
  - 29 strata
- inconsistency:
  - 9-13 stratum

Prioritising search: most versus least frequent



- Consider dividing s1 = {t,c,w}
- s1 occurs in 1a (note arbitrariness)
  - If s1 = {t} then 1a = torque or turkey
  - If s1 = {c} then 1a = clique
  - If s1 = {w} then 1a = wreath
- Frequency rank to obtain **ordered** sets:
  - {c, w, t} to try less frequent characters first
  - {t, c, w} to try more frequent characters first

Prioritising search: most versus least frequent first

Grid		Time (secs)			Arc-consistency (count)		
Words	Intersections	Default	Most	Least	Default	Most	Least
22	34	**14**	23	**14**	58	122	**51**
30	58	21	32	**19**	60	140	**43**
40	64	22	32	**18**	123	199	**69**
60	120	33	52	**31**	119	233	**68**

Proportion of least frequent

grid		1/2		1/4		1/8		1/16	
Words	Intersections	time	count	time	count	time	count	time	count
22	34	14	51	12	34	**10**	26	**10**	**23**
30	58	19	43	15	28	15	25	**13**	**21**
40	64	18	69	14	42	14	38	**12**	**29**
60	120	31	68	29	47	27	43	**24**	**37**

By product is some interesting words...